Laborator PS
Algoritmi de compresie a datelor

Lucrarea 1: Introducere în Compresia Datelor

Lucrarea 2: Algoritmul Shannon-Fano

Lucrarea 3: Algoritmul Huffman static

Lucrarea 4: Algoritmul Huffman dinamic

Prof. Dan Ştefănoiu <dan.stefanoiu@acse.pub.ro>

Ş.l. Alexandru Dumitraşcu <alexandru.dumitrascu@acse.pub.ro>

A. Obiective

Implementarea Algoritmului de compresie statică al lui Huffman. Testarea performanței acestuia, prin comparație cu cele ale Algoritmului Shannon-Fano şi ale programelor de uz general **WinZIP** şi **WinRAR**.

B. Suport teoretic

Metoda propusă de D.A. Huffman în 1952 păstrează o mare parte dintre caracteristicile Metodei Shannon-Fano. Mai precis, fazele de compresie şi decompresie trec prin aceleaşi etape. Diferența apare în maniera de construcție a arborelui binar, fapt care aduce o uşoară îmbunătățire a performanțelor de compresie. Continuăm să operăm cu alfabetul static de ordin 0.

În Metoda Shannon-Fano, arborele a fost construit plecând de la rădăcină (alfabetul $\mathcal{A}^0$) spre frunzele ocupate de simboli. Codul asociat unui simbol a fost construit dinspre bitul cel mai semnificativ spre bitul mai puțin semnificativ.

În Metoda Huffman, construcția arborelui pleacă de la frunze spre rădăcină. Aceasta implică o alcătuire a codurilor în sens invers, de la bitul cel mai puțin semnificativ, la bitul cel mai semnificativ.

Algoritmul de construcție a arborelui binar de tip Huffman este prezentat în continuare.

Algoritmul 3.1. *Construcția arborelui binar pentru compresia Huffman.*

1. Se parcurge setul de $N \in \mathbb{N}^*$ date în mod secvențial (simbol cu simbol) şi se construieşte alfabetul de ordin 0 ($\mathcal{A}^0$), contorizând numărul de apariții ale fiecărui simbol întâlnit. Astfel, fiecărui simbol $s \in \mathcal{A}^0$ i se asociază un contor $\mathcal{N}(s) \in \mathbb{N}^*$.

2. Se rearanjează simbolii alfabetului $\mathcal{A}^0$ în ordinea descrescătoare a numerelor de apariții:

$$\left[\begin{array}{l}\mathcal{A}^0=\{s_1,s_2,...,s_N\}\\ \mathcal{N}(s_1)\geq\mathcal{N}(s_2)\geq...\geq\mathcal{N}(s_N)\end{array}\right.$$

În cazul în care simboli diferiți au număre egale de apariții, ei vor fi ordonați lexicografic. Ordonarea alfabetului nu este, însă, strict necesară. Totuşi, ea este recomandată, pentru a aranja frunzele arborelui binar într-o manieră naturală şi, mai ales, pentru a permite o indexare mai uşoară a nodurilor intermediare ale acestuia. Ca să se înțeleagă mai bine maniera de construcție a arborelui binar, elementele alfabetului vor fi renotate (şi reindexate) astfel:

$$\mathcal{A}^0=\{t_{N+k-1}=s_k\}_{k\in\overline{1,N}}.$$

Simbolii t_N, t_{N+1},..., t_{2N-1} vor fi utilizați pentru a eticheta frunzele arborelui binar, iar alfabetul $\mathcal{A}^0$ devine *primul alfabet curent* al algoritmului şi va fi renotat prin $\mathcal{A}^{0,N}$. Alfabetul curent al algoritmului va fi notat prin $\mathcal{A}^{0,(N-k)}$, unde indicele k variază în gama $\overline{0,N-1}$. Aşadar, inițial, $k=0$ şi frunzele arborelui binar sunt etichetate cu simbolii primului alfabet curent.

3. Construcția arborelui binar se realizează simultan cu construcția noului alfabet curent $\mathcal{A}^{0,(N-k-1)}$ (unde $k\in\overline{0,N-2}$). Pentru a-l construi pe acesta din urmă din vechiul alfabet $\mathcal{A}^{0,(N-k)}$, se aplică următoarea regulă: se identifică 2 dintre simbolii cei mai *slabi* (adică având contoarele cele mai mici) din vechiul alfabet şi se elimină, înlocuindu-le cu un simbol virtual (notat ad hoc prin „Ø”), care are contorul egal cu *suma* contoarelor acestora; dacă există mai mult de 2 simboli slabi având acelaşi contor, se vor alege ultimii 2 întâlniți în alfabet. Nu mai este necesară reordonarea noului alfabet curent, astfel obținut. În plus, cardinalul său scade cu o unitate față de cel al alfabetului curent precedent. Aşadar:

$$\begin{cases} \mathcal{A}^{0,(N-k-1)} = \left[\mathcal{A}^{0,(N-k)} \setminus \{t_p, t_q\}\right] \cup \{t_{N-k-1}\} \\ t_{N-k-1} = \text{'Ø'} \text{ (simbol virtual)} \\ \mathcal{N}(t_{N-k-1}) = \mathcal{N}(t_p) + \mathcal{N}(t_q) \\ \#\mathcal{A}^{0,(N-k-1)} = \#\mathcal{A}^{0,(N-k)} - 1 = N - k - 1. \end{cases}.$$

Simbolul virtual astfel produs va eticheta un nod intermediar al arborelui binar, având ca descendenți cele două noduri etichetate de simbolii eliminați. Cei 2 simboli eliminați (t_p şi t_q) etichetează:

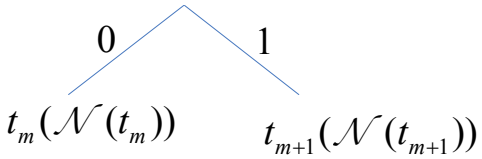
- două frunze al arborelui, dacă aparțin alfabetului $\mathcal{A}^{0,N}$;
- un nod intermediar al arborelui şi o frunză, dacă doar unul dintre ei aparține alfabetului $\mathcal{A}^{0,N}$;
- două noduri intermediare ale arborelui, dacă nici unul dintre ei nu aparține alfabetului $\mathcal{A}^{0,N}$.

În acest fel, fiecare nod intermediar al arborelui posedă câte doi descendenți direcți (*fii*), iar contorul fiului din stânga este cel puțin egal cu cel al fiului din dreapta. Arcele de legătură vor fi etichetate cu 0 (pentru fiul din stânga) şi cu 1 (pentru fiul din dreapta). Aceste etichete reprezintă, de fapt, valorile biților noilor coduri asociate simbolilor.

4. Se repetă pasul anterior pentru fiecare alfabet curent $\mathcal{A}^{0,(N-k-1)}$ (unde $k \in \overline{0, N-2}$), adică de $N-1$ ori, până când arborelui binar i se stabileşte rădăcina (t_1).
5. Se reindexează nodurile arborelui astfel încât fiii oricărui nod care posedă descendenți să fie etichetați cu indici consecutivi (de la stânga la dreapta). (Reindexarea nodurilor nu este o operație absolut necesară, dar este utilă în faza de decompresie.)

Regula de bază pentru structura arborelui binar construit cu ajutorul acestui algoritm este cea a adăugării de noi noduri intermediare prin însumarea ponderilor simbolilor celor mai slabi, avansând de la frunze spre rădăcină. De exemplu, în arborele final (cu noduri reindexate), nodul notat cu t_n are 2 fii: t_m şi t_{m+1}, ca în imaginea următoare:

$$t_n(\mathcal{N}(t_n)) = t_n(\mathcal{N}(t_m)) + t_n(\mathcal{N}(t_{m+1}))$$



Aşa cum se constată, fiecare nod al arborelui binar este caracterizat de două informații: o *etichetă* (de tip t_n) (care dă numele nodului şi îl indexează) şi o *pondere* (contorul $\mathcal{N}(t_n)$). Nodurile sunt indexate în ordinea descrescătoare a ponderilor lor, începând de la rădăcină spre frunze, datorită ultimului pas al algoritmului.

Rădăcina (t_1) deține ponderea maximă, deoarece aceasta este egală cu suma tuturor contoarelor (adică volumul setului de date). Arborele are, în final, $2N-1$ noduri şi construcția sa se efectuează în $N-1$ paşi. Fiecare nod va avea asociat un cod unic, obținut prin concatenarea valorilor (0 sau 1 ale) arcelor ce unesc rădăcina cu acel nod.

Exemplu

Se revine la exemplul din cadrul secțiunii precedente (Metoda Shannon-Fano):

$$\mathcal{D}: \text{IT\_IS\_BETTER\_LATER\_THAN\_NEVER.}$$

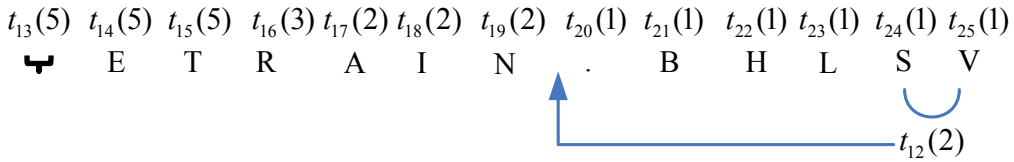
Pentru acesta, se va construi mai întîi arborele binar de tip Huffman. Apoi se va enumera şirul de biți ai datelor comprimate şi, în final, se va indica maniera de decompresie.

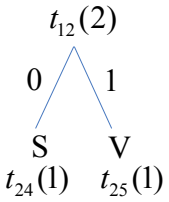
Alfabetul asociat setului de date $\mathcal{D}$ (de mai sus) are $N=13$ elemente, astfel că arborele final va avea 25 de noduri. Alfabetul poate fi scris în manieră ordonată ca în tabelul următor (cu atribuirea inițială a etichetelor frunzelor arborelui binar):

Nod	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}
$\mathcal{A}^{0,13}$	␣	E	T	R	A	I	N	.	B	H	L	S	V
$\mathcal{N}(s)$	5	5	5	3	2	2	2	1	1	1	1	1	1

Vom parcurge Algoritmul 3.1, indicând fiecare pas prin valoarea indicelui k. Vor fi 12 paşi. Pentru primii 10 paşi, ilustrăm structura arborelui parțial şi forma alfabetului curent (indicând prin caractere aldine simbolul virtual înlocuitor). Datorită simplității lor, ultimii 2 paşi au fost compactați, doar structura arborelui final fiind reprezentată (cu nodurile reindexate).

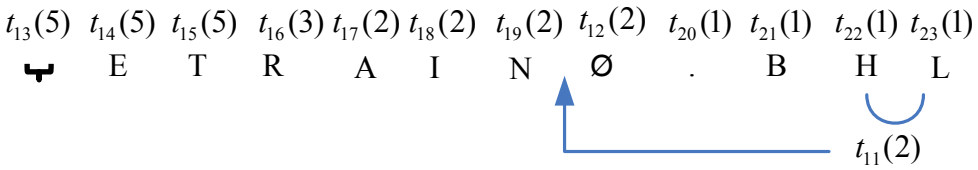
$k=1$:

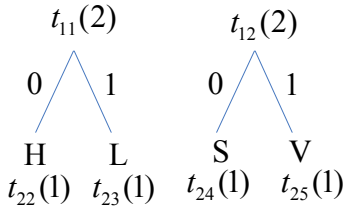




Nod	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{12}	t_{20}	t_{21}	t_{22}	t_{23}
$\mathcal{A}^{0,12}$	␣	E	T	R	A	I	N	Ø	.	B	H	L
$\mathcal{N}(s)$	5	5	5	3	2	2	2	2	1	1	1	1

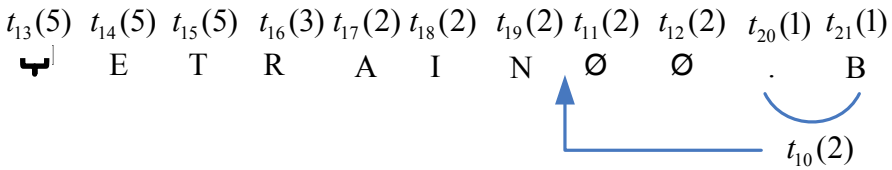
$k=2$:

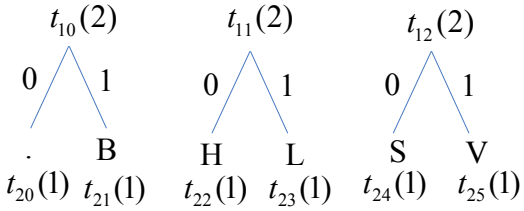




Nod	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{11}	t_{12}	t_{20}	t_{21}
$\mathcal{A}^{0,11}$	␣	E	T	R	A	I	N	Ø	Ø	.	B
$\mathcal{N}(s)$	5	5	5	3	2	2	2	2	2	1	1

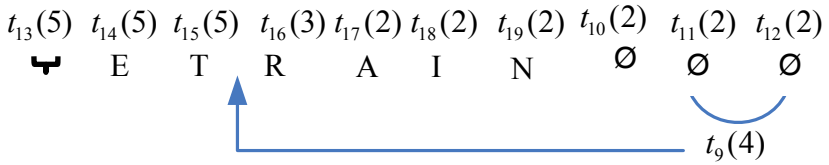
$k = 3$:

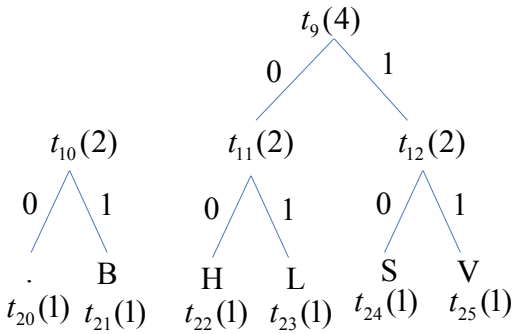




Nod	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{10}	t_{11}	t_{12}
$\mathcal{A}^{0,10}$	␣	E	T	R	A	I	N	Ø	Ø	Ø
$\mathcal{N}(s)$	5	5	5	3	2	2	2	2	2	2

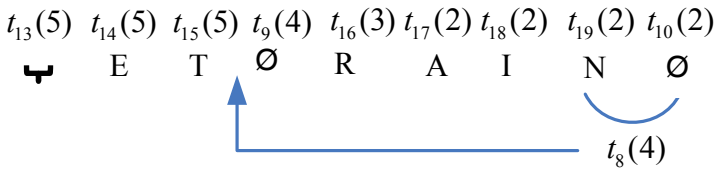
$k = 4$:

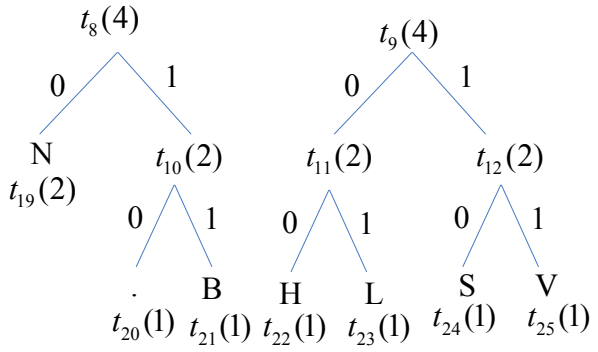




Nod	t_{13}	t_{14}	t_{15}	t_9	t_{16}	t_{17}	t_{18}	t_{19}	t_{10}
$\mathcal{A}^{0,9}$	␣	E	T	Ø	R	A	I	N	Ø
$\mathcal{N}(s)$	5	5	5	4	3	2	2	2	2

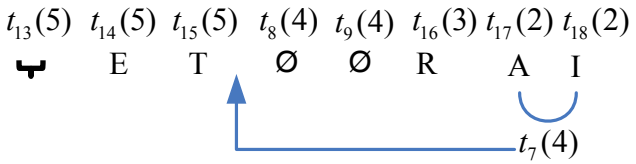
$k = 5$:

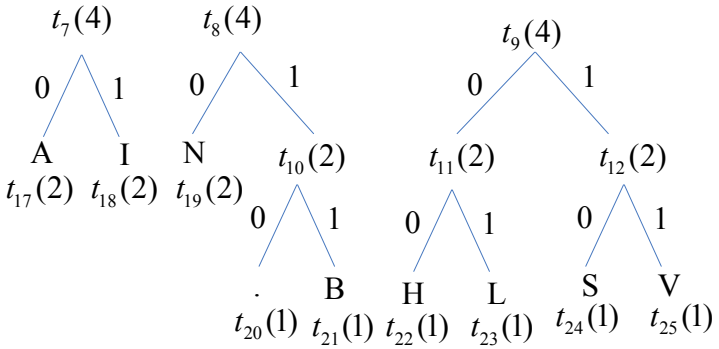




Nod	t_{13}	t_{14}	t_{15}	t_8	t_9	t_{16}	t_{17}	t_{18}
$\mathcal{A}^{0,8}$	␣	E	T	Ø	Ø	R	A	I
$\mathcal{N}(s)$	5	5	5	4	4	3	2	2

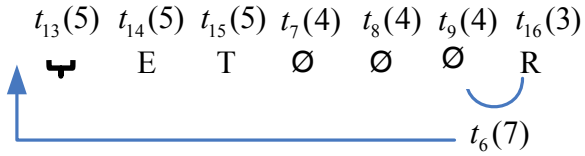
$k = 6$:

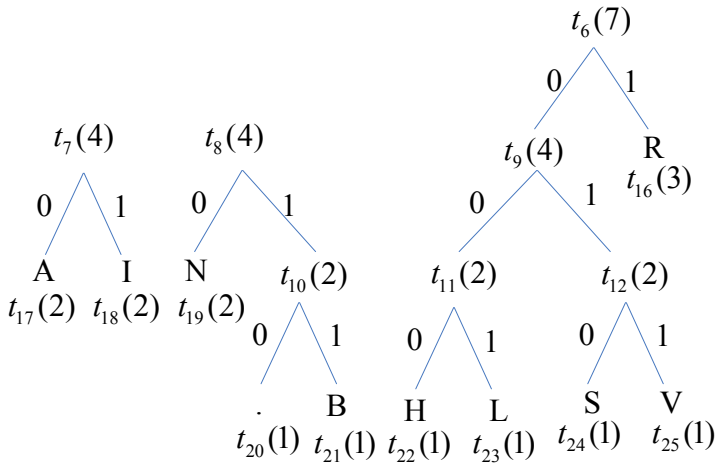




Nod	t_{13}	t_{14}	t_{15}	t_7	t_8	t_9	t_{16}
$\mathcal{A}^{0,7}$	␣	E	T	Ø	Ø	Ø	R
$\mathcal{N}(s)$	5	5	5	4	4	4	3

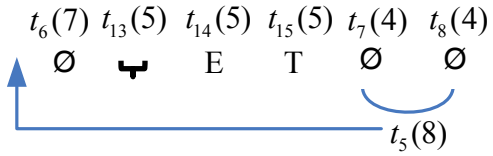
$k = 7$:

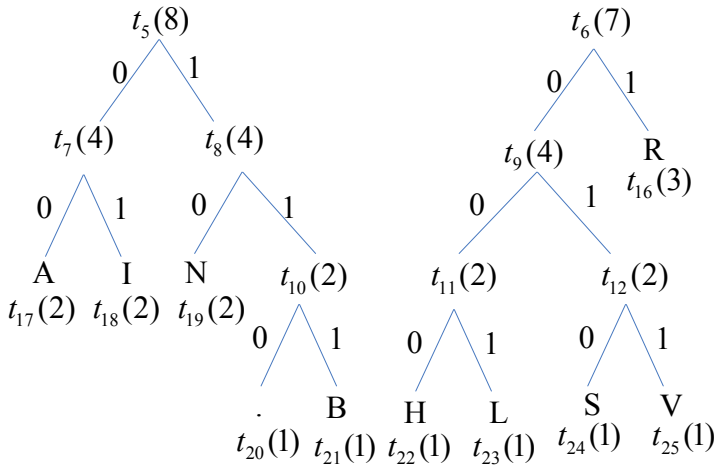




Nod	t_6	t_{13}	t_{14}	t_{15}	t_7	t_8
$\mathcal{A}^{0,6}$	Ø	␣	E	T	Ø	Ø
$\mathcal{N}(s)$	7	5	5	5	4	4

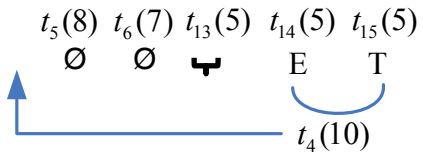
$k=8$:

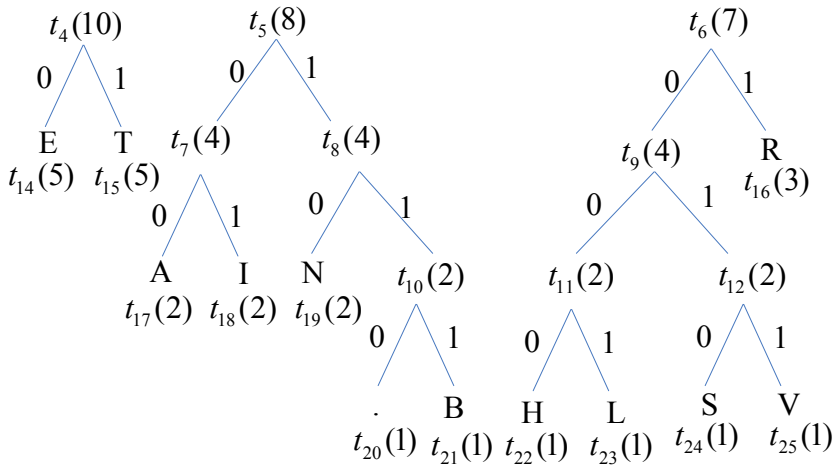




Nod	t_5	t_6	t_{13}	t_{14}	t_{15}
$\mathcal{A}^{0,5}$	Ø	Ø	␣	E	T
$\mathcal{N}(s)$	8	7	5	5	5

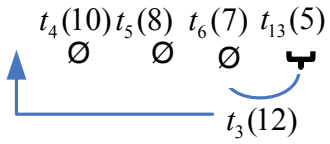
$k = 9$:

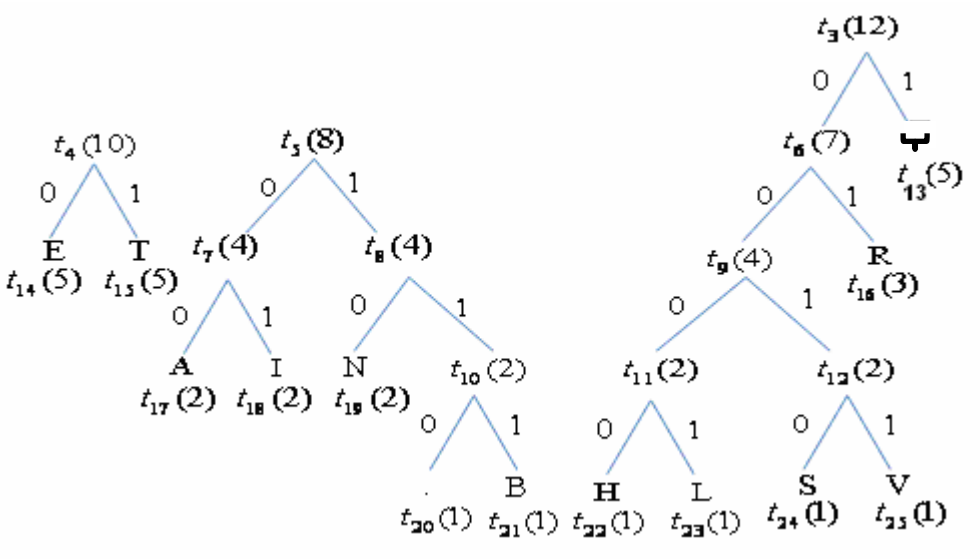




Nod	t_4	t_5	t_6	t_{13}
$\mathcal{A}^{0,4}$	Ø	Ø	Ø	␣
$\mathcal{N}(s)$	10	8	7	5

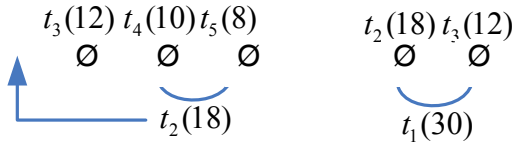
$k=10$:

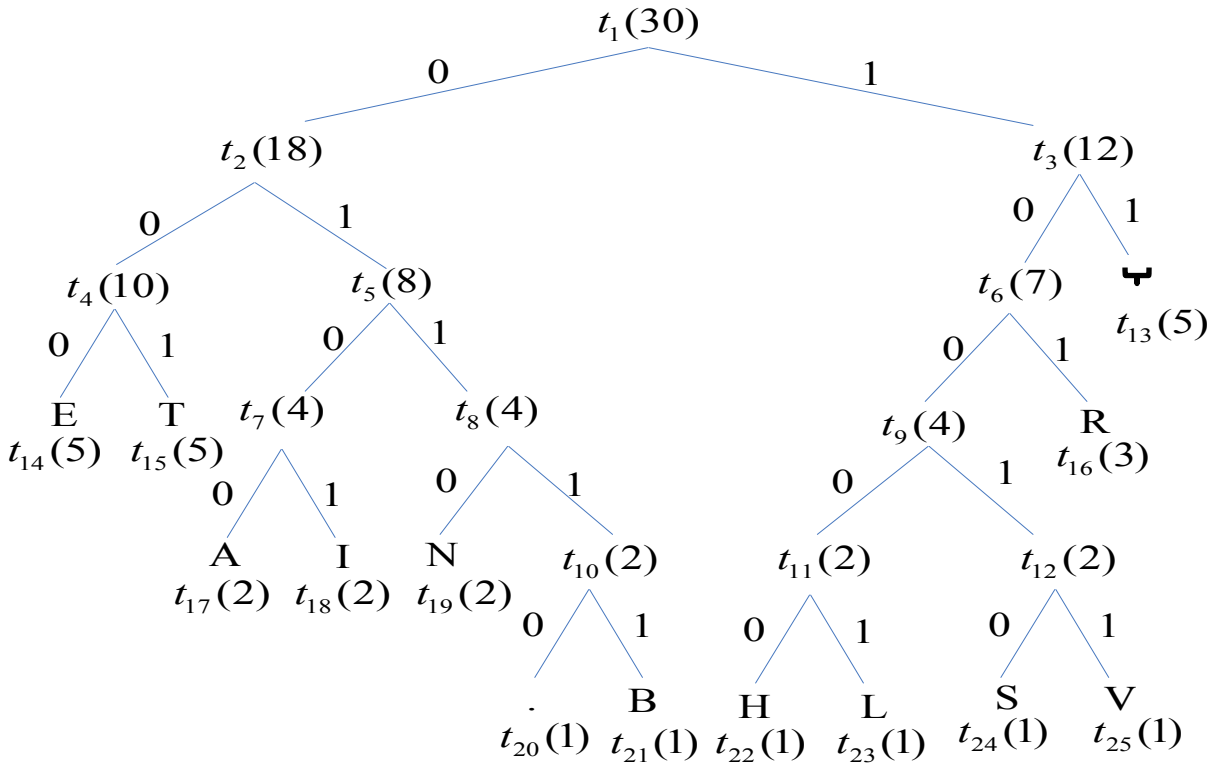




Nod	t_3	t_4	t_5
$\mathcal{A}^{0,3}$	Ø	Ø	Ø
$\mathcal{N}(s)$	12	10	8

$k \in \{11,12\}$:





Informația înscrisă pe fluxul de ieşire, corespunzătoare arborelui binar construit anterior, va arăta ca în Tabelele 3.1 şi 3.2.

Tabelul 3.1. *Informație auxiliară statică de tip Huffman.*

Cod	13	32	5	69	5	84	5	82	3	65	2	72	2	78	2	46	1	66	1	72	1	76	1	83	1	86
Info	N	␣	$\mathcal{N}$	E	$\mathcal{N}$	T	$\mathcal{N}$	R	$\mathcal{N}$	A	$\mathcal{N}$	I	$\mathcal{N}$	N	$\mathcal{N}$	.	$\mathcal{N}$	B	$\mathcal{N}$	H	$\mathcal{N}$	L	$\mathcal{N}$	S	$\mathcal{N}$	V
Nr. biți	8	8	32	8	32	8	32	8	32	8	32	8	32	8	32	8	32	8	32	8	32	8	32	8	32	8

Tabelul 3.2. *Informație utilă statică de tip Huffman.*

Cod	0101	001	11	0101	10010	11	01111	000	001	001	000	101	11	10001	1100
Info	I	T	␣	I	S	␣	B	E	T	T	E	R	␣	L	A
Nr. biți	4	3	2	4	5	2	5	3	3	3	3	3	2	5	4
Cod	001	000	101	11	001	10000	1000	0110	11	0110	000	10011	000	101	01110
Info	T	E	R	␣	T	H	A	N	␣	N	E	V	E	R	.
Nr. biți	3	3	3	2	3	5	4	4	2	4	3	5	3	3	5

Practic, volumul de date comprimate nu s-a schimbat, în acest exemplu, față de cel din lucrarea precedentă. Doar noile coduri utile sunt altele. Însă, în cazul multor seturi de date (sensibil mai lungi decât acesta), Metoda Huffman este mai eficientă decât Metoda Shannon-Fano.

Şi decompresia datelor se desfăşoară ca în cazul Metodei Shannon-Fano. Semnalăm totuşi faptul că, în cazul implementării celor două metode pe un mijloc automat de calcul, arborele construit prin Metoda Huffman este mai uşor de reprezentat şi de construit decât arborele Shannon-Fano, programul având o complexitate uşor mai redusă.

C. Sarcini de lucru

💻 **Tema 1 (Implementarea algoritmilor Huffman statici)**

a. Scrieți 3 rutine corespunzătoare Algoritmului 3.1, şi metodelor de compresie-decompresie din Metoda Huffman statică.

b. Verificați corectitudinea lor cu ajutorul unui program de test aplicat exemplului din subsecțiunea precedentă (textul trebuie să fie reconstituit perfect).

c. Evaluați factorul de compresie realizat în acest exemplu (ținînd cont şi de informația auxiliară). Este capabilă metoda să realizeze compresia?

💻 **Tema 2 (Testarea algoritmilor Huffman statici)**

a. Scrieți un program care realizează compresia şi decompresia celor 12 fişiere din corpusul de date descris în Lucrarea 1, cu ajutorul rutinelor de la tema precedentă.

b. Calculați norma diferenței dintre setul inițial de date şi cel decomprimat. Pentru fiecare set de date, indicați factorul de compresie aferent (incluzând informația auxiliară) şi evaluați duratele de compresie, respectiv decompresie. Comentați rezultatul obținut, cu referire la capacitatea de compresie a metodei pentru diferite tipuri de fişiere.

c. Comparați factorii de compresie de la punctul precedent cu cei obținuți prin utilizarea Metodei Shannon-Fano şi a celor două programe de uz general **WinZIP** şi **WinRAR**. (Utilizați funcția **fread** pentru a evalua exact numărul de biți ai arhivelor produse de aceste utilitare.) Comentați rezultatele obținute, cu referire la compromisul dintre complexitatea algoritmilor de compresie şi performanțele acestora.